Victorian 6502 User Group Newsletter

KAOS

For People Who Have Got Smart

OSI SYM KIM AIM APPLE UK101 ORANGE

Registered by Australia Post
Publication No. VBG4212

ADDRESS ALL CORRESPONDENCE TO 10 FORBES ST ESSENDON, VIC. 3040

Vol.3 No.9 June 1983

We were recently contacted by David Lutz from The Australian Beginning (T.A.B.) who had all sorts of good things to tell us. The most important being that the T.A.B. is under new management and that their policy is to encourage more users by keeping charges as low as possible. Joining fees will be reduced and anyone who joins through a user group will be regarded as a charter member and will get a 20% discount on user charges. We have also settled the matter of the $5 credit owed to KAOS members who paid $40 to join the T.A.B., if they contact David Lutz at 24 Camberwell Rd, Hawthorn East, 3123 ph.03 813 1133 and give him their user name, he will ensure that they receive a $5 credit.

For interstate members, T.A.B. hopes to be on AUSTPAC by the end of July and we have been told that AUSTPAC will be free for the first six months, after that we believe the charge will be a local call fee plus $3 per hour. There is a new and much enlarged manual available, incorporating all changes made up to date, a "Chat Mode" which allows 5 users at a time to contact each other via their keyboard and screen. The T.A.B. will also let us have the protocol for their down-loading program to make it possible for someone in the club to write a down-loading program for the OSI. These are only some of the changes being made so all and all things are definitely looking up. We hope to have an article from David Lutz for our next newsletter giving more details of what is happening.

The next meeting will be on Sunday 26th June at 2pm at the Essendon Primary School which is on the corner of Raleigh and Nicholson Streets, Essendon. The school will be open for members from 1pm.

The closing date for articles for the July newsletter will be Friday 15th July.

INDEX

THE BEGINNING MACHINE LANGUAGE PROGRAMMER.......Part 12

by David Dodds

Last month in BMLP we left off in the middle of the development of a suite of subroutines to control the movement of the cursor of ARTIST. Each of the four routines calculates a new cursor position and then calls UPDATE, a routine which has the task of validating the proposed new cursor position.

The tasks to be performed by UPDATE are fairly straightforward :

1 Check if the cursor is off the top of the screen
2 Check if the cursor is off the bottom of the screen

and if on the screen

3 Restore the character under the cursor (the cursor character might be on the screen at the time)
4 Update the cursor position
5 Save the character under the new cursor position.

In BASIC this might be accomplished by a line such as:

```
IF NEWCUR > TV.TOP AND NEWCUR < TV.END THEN GOSUB.....
```

In ARTIST the details of the start and end of screen memory are initialised into two locations known as TV.TOP and TV.END. In determining whether the cursor is off the top of the screen, as the 6502 can only handle 8 bits at a time it is necessary to do the test in 2 parts.
Firstly compare the high order bytes of NEWCUR and TV.TOP, if NEWCUR+1 > TV.TOP+1 then no further testing is required. If NEWCUR+1 = TV.TOP+1 then compare the low order bytes. When NEWCUR is > or = TV.TOP then the cursor may be updated.

The test for off the bottom of the screen is easier because increasing the value of the cursor by 1 will cause the NEWCUR+1 to become greater than TV.END+1 so that testing of the high byte only is required.

This sequence in assembly language becomes:

```
UPDATE LDA NEWCUR+1     ;test high order byte
       CMP TV.TOP+1     ;of new cursor position
       BEQ UD1          ;test further if equal
       BCC FAIL
       BCS BOTPGE       ;carry set if newcur >or=
UD1    LDA NEWCUR       ;test low byte of new cursor
       CMP TV.TOP
       BCS NU2OLD       ;carry set if newcur > or =
       BCC FAIL
BOTPGE LDA TV.END+1     ;test off bottom of screen
       CMP NEWCUR+1
       BCC FAIL
NU2OLD LDA TVTEMP       ;restore charcter to screen
       LDY #0
       STA (CURPOS),Y
       LDA NEWCUR       ;update cursor position
       LDX NEWCUR+1
       STA CURPOS
       STX CURPOS+1
       LDA (CURPOS),Y ;save new cursor character
       STA TVTEMP
FAIL   RTS
```

Note:- TVTEMP -where the current character under the cursor is stored
CURPOS -current cursor position
NEWCUR -new cursor position

MORE FROM THE WEST

by Wayne Geary

The revised version of the Word Processor that I wrote of in my last article (More from the West in Vol.3 No.7) although now residing in about 110 bytes less than the original version still requires 108 bytes in excess of 2K to reside in EPROM. This can be accomodated in one of three ways:-
1. Use of a consecutive block of memory.
2. Placing the tables which require 130 bytes elsewhere in memory where there is a space.
3. Placing the tables and Menu jumplist combined into a separate area where there is 208 bytes spare.

The revised version now only uses page-0 addresses from $00 to $A2, ie. about half of the original requirements. Also included are some mods to improve the various capabilities, for which thanks to Bert Patterson for the suggestions.

Finally I have an Assembler source code for the revised version so that for those wishing to burn the Word Processor into EPROM I can provide a cassette dump of a version to suit (almost) anyones requirements if they specify their EPROM buffer area (to be less than $7FFF) and where exactly they wish the Word Processor to reside in their system. Cost of cassette dump would be $10 to cover the cost of the cassette and postage etc.

I have just completed a Speech Synthesis Board based on the Votrax SC-01 speech chip which is compatible with the 48 pin OSI bus of the C2/C3/C4 systems and should be fairly easy to connect to a C1/Superboard.

The board has both address and data buffers and uses port A of a PIA located at page $F7XX to control the SC-01 speech chip. Port B is taken to an edge connector for user expansion along with the SC-01 audio output. The last item on the board is a socket for a 2732 4K EPROM located at $CXXX but fairly easily deleted or relocated to another area of memory.

The board is about 20cm by 10cm, is single sided, tinned and drilled, and requires 46 links. Cost of the board including documentation and postagge is $25.

At the above price anyone should be able to add speech to their OSI for about $150 which compares well with Dick Smith's Type and Talk for around $500.

I have written an initialisation routine for the PIA and a small routine to drive the SC-01 which I can provide and intend to try(!!) and write my own Type -Talk algorithm which will go into the 2732 on the board. (Any takers for some help here?)

My only request is that anyone who is interested write or order NOW, and not some time next year. Anyone who is interested can write to me at:

HELP

A call for help comes from Kev Goffey, he has an IDS Paper Tiger 4456 printer with graphics option. Kev has not yet made use of the graphics option and is wondering if there is any other member with the same or similar printer with whom he can compare notes. Kev's address is:

Superboard

June 1983

Newsletter of the Ohio Superboard User Group, 146 York Street, Nundah, 4012.

EPROM PROGRAMMER TO SUIT RABBLE BOARD by *Bernie Wills.*

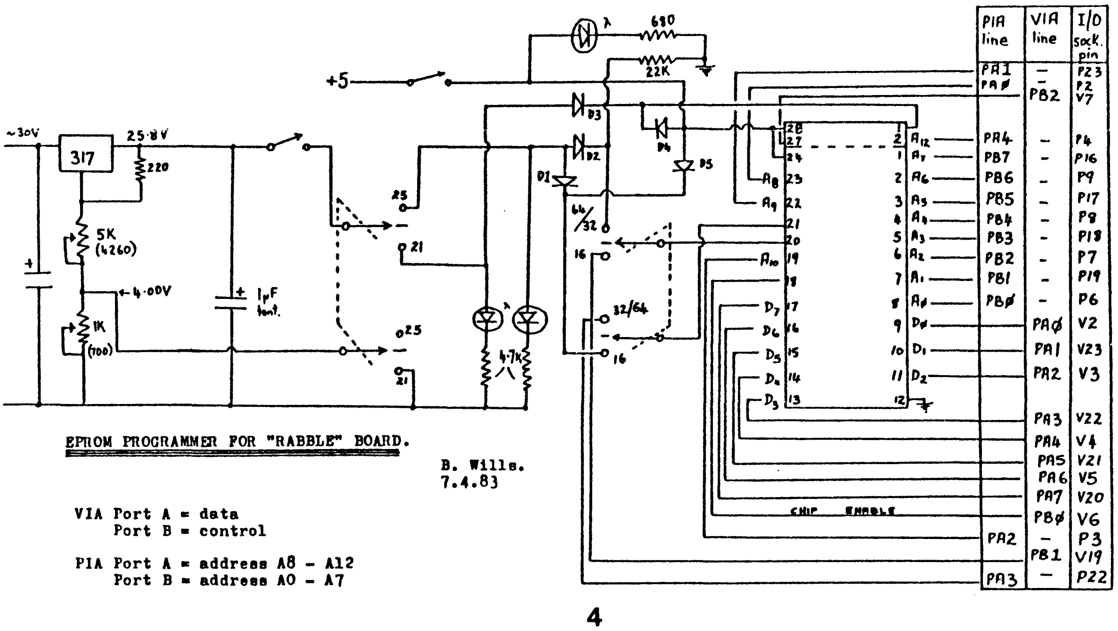
Design: The circuit uses the I/O sockets of the PIA and VIA on the Rabble board. The PIA, which requires more setting up instructions, is used to generate addresses, and both ports are set for output. Port B = A0 - A7 and Port A = A8 - A12. The VIA is used for control signals on port B, and for data in and out on port A. The circuit is designed for single rail +5 eproms but will not suit 2532 or 2764 with different pin usage. If you must program these types, then a personality socket design would be needed, as well as changes to the software. The circuit has been tried for 2716 and 2732, but not with a 2764. Wiring for the 2764 was based on an article in ETI magazine, February, 1983. It was assumed that this type was available from suppliers.

Parts: With the exception of the programming power supply, some diodes and toggle switches, and a 28 pin zero insertion force (ZIF) socket, the major components required to make up this programmer are to be found on the Rabble expansion board.

Power: A 30 volt unregulated supply was used to derive the programming voltages of 25 and 21 volts. The 5 volt line was taken from the fused supply on the Superboard. A ground or common connection must also be supplied.

Software: The software to drive the programmer is in Basic and is too long to reproduce in this newsletter. Two versions are available, one being well documented with REMs and the second a compacted version to use for the actual programming. The software covers all functions associated with Eprom programming. The program assumes the data will be at $3ØØØ for a 2732 and at $38ØØ for a 2716.

More Info: More information and a full sized circuit diagram may be requested from the User Group by sending a SAE. If the software is required, send us a cassette and include your address and one 27¢ stamp and one 40¢ stamp to cover the postage costs. More information will be included with your cassette.



PIA line	VIA line	I/O sock. pin
PA1	–	P23
PAØ	–	P2
	PB2	V7
PA4	-	P4
PB7	-	P16
PB6	-	P9
PB5	-	P17
PB4	-	P8
PB3	-	P18
PB2	-	P7
PB1	-	P19
PBØ	-	P6
	PAØ	V2
	PA1	V23
	PA2	V3
	PA3	V22
	PA4	V4
	PA5	V21
	PA6	V5
	PA7	V20
	PBØ	V6
PA2	–	P3
	PB1	V19
PA3	–	P22

EPROM PROGRAMMER FOR "RABBLE" BOARD.

B. Wills.
7.4.83

VIA Port A = data
Port B = control

PIA Port A = address A8 - A12
Port B = address A0 - A7

— SUPERBOARD —

SOFTWARE REVIEW : Orbital Lander.

Orbital Lander is a rather quaint little Basic game of about 3k bytes. You control a rocket (lander) with a shift key. The direction of travel is vertical only. The rocket starts in mid-air, and you first have to bring down the rocket to gain extra fuel. Then you take off, and attempt to rendezvous with an orbiter which travels continuously from left to right across the screen, and at various altitudes.

The rocket behaves as you would expect,and doesn't stop moving immediately when you burn fuel, or cease to do so. You get to safely land,then dock with the orbiter, or bust up to ten rockets in the attempt. The program then comments on your skill (or lack of it).

It's all rather boring after the first couple of goes, and you don't even get a spectacular crash when you fail to dock and, out of fuel, plummet to the surface.
Orbital Lander is an Aardvark program, and is in the OSUG Library.

Bernie Wills has changed his Basic 1 Eprom so that the line feed after printing the OK message is prevented. This is a big improvement in the 12 x 48 mode.

```
Original code:  $A192 OD OA 4F 4B OD OA 00
New code     :  $A192 OD OA 20 4F 4B OD 00
```

The prompt now becomes _OK, and when you start typing, you overwrite OK.

NEATLY DOCUMENTING PROGRAMS by Bob Ellis

It is truly amazing that you can play around with a computer for years without ever finding out everything about it. For example, Microsoft Basic will support an alpha string after a GOSUB or GOTO, making for neat documentation. This program will run:- .

```
10 PRINT"TEST": GOSUB 30 PRINT 1-10: PRINT"END"
20 GOTO 40 END PROGRAM
30 FOR R=1 TO 10: PRINT R;:NEXT:RETURN
40 END
```

SUSPECT OLD ADAGES by Ken O'Rourke

Basic programs run faster if you put all the statements on one line using colons! Big line numbers make programs run slower! True, or false?
Both statements are true, but the difference is insignificant. Try it!
Much more important is what you put after NEXT.

```
10 FOR R=1 TO 1000:FOR X=1 TO 60:NEXT X,R    takes 77 secs at 1MHz clock.
20 FOR R=1 TO 1000:FOR X=1 TO 60:NEXT:NEXT   takes 61 seconds, 20% faster.
```

What about defining the most used variable, X. Adding line 9 X=0 makes for a 3 second improvement with line 10 and no change with line 20 (10 deleted) If we define the other two numbers as variables as below, we gain 2 seconds.

```
 9 Z=60:Y=1000
10 FOR R=1 TO Y:FOR X=1 TO Z:NEXT:NEXT   takes 59 seconds.
```

A situation where you do get a big change is when poking to a screen in a game.

```
10 X=54000:Y=0:FOR R=1 TO 10000:POKE X,Y:NEXT    takes 26 seconds.
10 FOR R=1 TO 10000:POKE 54000,0:NEXT            takes 65 seconds.
```

Does GOSUB work faster when the routine is in early line numbers, or when it is as near as possible after the line which calls it? Try it!

Ed Richardson.

MY SUPERBOARD II SERIES 2. Part 3
by John Whitehead

The Stack, Non-maskable interrupt and Real Time clock.

The 6502 stack is in page one ($0100 to $01FF) and is used by the processsor as a last in first out temporary store for subroutine return addresses, etc.

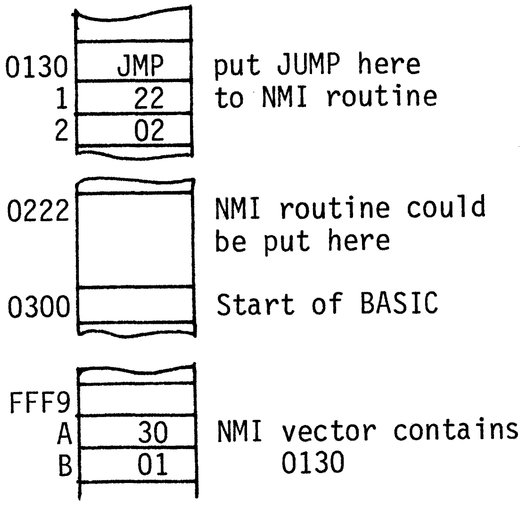
The stack is initialized by putting its start address into the 6502 Stack pointer register. The stack grows downward when data is Pushed into it, and the stack pointer is decremented to point to the bottom of the stack. The stack goes up when data is removed. The Monitor ROM (65V) initializes the stack to $0128, BASIC to $01FC and Exmon to $01FF.

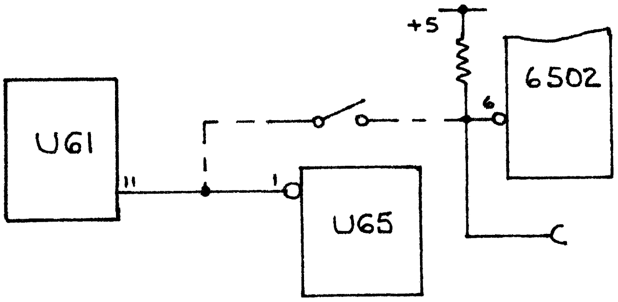
The Non-Maskable Interupt (NMI) is a hardware input to the 6502 on pin 6. When this pin is driven low the 6502 stops running its present program, loads the program counter and processor status registers onto the stack for later use, then puts the contents of $FFFA and $FFFB (which is $0130) in the program counter. The 6502 then Jumps to $0130 and runs the program found there. This program ends with an RTI (return from interrupt), and the 6502 then goes back to the original program.

The Real time clock hardware is simply a single pole on/off switch that connects the vertical sync clock pulse C15 to the NMI pin of the 6502. The software displays the time at the bottom of the screen below the guardband to prevent it scrolling up the screen. The original program in KAOS July 81 contained a JMP $0000 which caused problems when running machine code programs. I have removed this and added an alarm.display.

Due to the NMI vector at $0130 being in the stack (which can only be fixed by changing the Monitor EPROM), the computer will stop running if the stack grows down to $0132 and changes the interupt program pointers, but this does not normally happen.

I have the clock program in EPROM at $E000 and it uses my scratch pad RAM at $CC00 to store the time. The BASIC program below puts the program into page two from $0222 onwards.





```
50000 PRINTCHR$(127)"Real time clock
50010 PRINT"modified with alarm by John Whitehead
50020 PRINT:INPUT"Set time(hr,min)";H,M
50030 GOSUB50250:POKE736,H:POKE737,M:POKE738,0
50040 PRINT:INPUT"Set alarm time(hr,min)";H,M
50050 GOSUB50250:POKE740,H:POKE741,M:POKE742,0:POKE743,255
50060 POKE304,76:POKE305,34:POKE306,2
50070 DATAZZZ:READA$:IFA$<>"ZZZ"THEN50070
50080 :
50090 DATA 72,206,227,2,208,112,138,72,152,72,248,169,60
50100 DATA 141,227,2,162,2,24,189,224,2,105,1,157,224,2
50110 DATA 202,48,11,201,96,208,16,169,0,157,225,2,240
50120 DATA 233,201,19,208,5,169,1,141,224,2,162,2,160,8
50130 DATA 189,224,2,32,159,2,169,58,153,238,211,136,202
50140 DATA 16,241,162,2,189,228,2,221,224,2,208,6,202,16
50150 DATA 245,238,231,2,173,231,2,208,24,162,0,189,154
50160 DATA 2,157,230,211,232,224,5,208,245,173,226,2,106
50170 DATA 144,5,169,65,141,230,211,104,168,104,170,104
50180 DATA 64,97,108,97,114,109,32,166,2,74,74,74,74,72
50190 DATA 41,15,9,48,153,238,211,136,104,96,32,238,255
50200 DATA 32,235,255,32,238,255,56,233,48,10,10,10,10
50210 DATA 141,227,2,32,235,255,32,238,255,56,233,48,24
50220 DATA 109,227,2,96,255,255,255,255,255,255,255,255
50230 FORC=0TO183:READI:POKE 546 +C,I:NEXT
50240 PRINT:PRINT"Turn on NMI switch":END
50250 IFH>12ORM>59ORS>59THENPRINT"Illegal time":GOTO50020
50260 H=INT(H/10)*6+H:M=INT(M/10)*6+M:RETURN
```

THE OHIO PRINTER STANDARD ON OS65D

by Michael Lemaire

There has been a fair amount of dissention over where to put the printer port on Ohio disk machines - basically, should it be device #1 or #4.

According to the Ohio standard, the printer should be run by a parallel interface (Centronics), implemented with a PIA, on device #4. Devive #1 is meant to be a serial port to a terminal (eg. on C3's); alternativly, on a memory mapped video system, it can be a cassette interface or modem.

The reason for adhering to such a standard is to facilitate the transfer of software between machines - this is why CP/M is popular; not because of any good features, but because of the vast supply of (or market for) software for any machine running CP/M.

Implementing the device #4 printer driver:

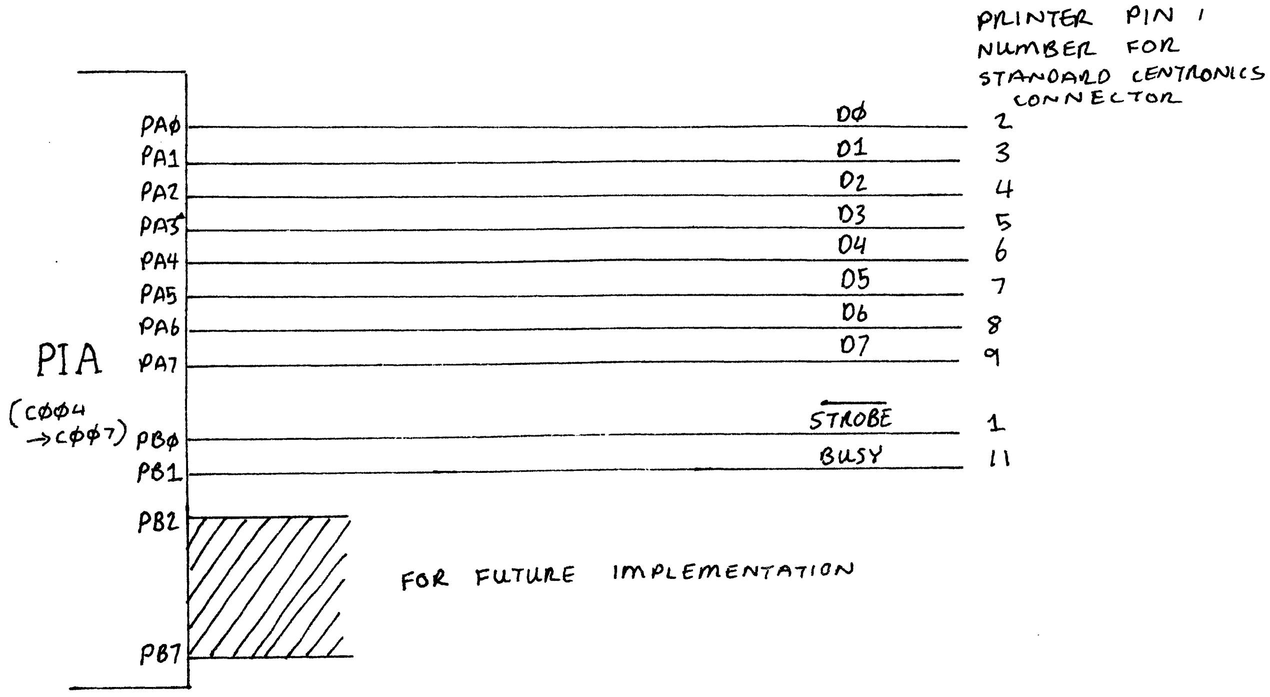
Output routine for OS65D:

```
             * = $249F
            STA   $C004    send char to printer,
            DEC   $C006    strobe printer
            INC   $C006
     LOOP   LDA   $C006    check busy
            AND   #02
            BNE   LOOP
            RTS
```

PIA initialization in BEXEC*

```
P=49156:POKE P,255:POKE P+1,4:POKE P+2,1:POKE P+3,4:POKE P+2,1
```

Next page please



AN IMPROVED KEYBOARD ALGORITHM Pt2

by Rodney Eisfelder

As promised last month here is a listing of the new, improved keyboard routine.

```
100;       REVISED KEYBOARD ROUTINE
120;       REPLACES ROM ROUTINE AND IS DESIGNED TO
140;       BEHAVE LIKE A NORMAL COMPUTER TERMINAL
160;       IN THAT SHIFT LOCK ONLY AFFECTS ALPHA KEYS
180;
200;
220; R.EISFELDER DEC 5 1982
240;
280;       VERSION MAY24 1983
260;
300;       SPECIFICATIONS ARE:
320;
340; SLk LS RS AL NO SP
360;  0   0  0 LC NM NM
380:  0   0  1 US SH NM
400;  0   1  0 UC SH NM
420;  0   1  1 UC SH NM
440;
460;  1   0  0 UC NM NM
480;  1   0  1 US SH NM
500;  1   1  0 US SH NM
520;  1   1  1 US SH NM
540;
560; WHERE SLk MEANS SHIFT LOCK
580; LS/RS MEANS LEFT/RIGHT SHIFT
600; AL MEANS A-Z
620; NO MEANS 0-9 AND PUNCTUATION (;,:-./)
640; SP MEANS ESC,LF,CR,RUBOUT,SPACE
660; LC MEANS LOWER CASE LETTER
680; UC MEANS UPPER CASE LETTERS
700; US MEANS UPPER CASE AND SPECIAL (SHIFT K TO P)
720; NM MEANS NORMAL (THE LOWER CHARACTER ON THE KEY)
740; SH MEANS SHIFTED (THE UPPER CHARACTER ON THE KEY)
760;
```

```
 780; WE CAN REUSE THE PUT ROW AND GET COLUMN
 800; ROUTINES
 820PUTROW = $FCBE ;ON C1P
 840GETCOL = $FCC6 ;ON C1P - PRESERVES A&Y,RESULT IN X
 860GETCL2 = $FCCF ;ON C1P - RESULT IN A, X&Y PRESERVED
 880;
 900; FOR THE DISK VERSION, IT WOULD BE BETTER
 920; IF THE FOLLOWING FOUR VARIABLES WERE
 940; PUT SOMEWHERE ELSE, BUT WE WILL LEAVE THEM
 960; ON PAGE 2 FOR COMPATABILITY WITH THE
 980; EXISTING ROUTINE
1000;
1020RESULT = $213
1040ITCNT = $214
1060RAWKEY = $215
1080OLDKEY = $216
1100BITNUM = $FDC8  ;REUSE THIS ROUTINE
1120;
1140; REUSE THE TABLE OF CHARACTER VALUES.
1160; NOTE THAT THE TOP BIT OF EACH CHARACTER
1180; IS IGNORED IN THIS VERSION
1200;
1220TABLE = $FDCF
1240; WE SAVE FIVE BYTES BY USING A DELAY ROUTINE
1260; PROVIDED FOR THE DISK BOOT
1280W1250U = $FC91  ;WAIT 1250*X MICRO SECONDS
1300;
1320; THE TECHNIQUE OF HIDING INSTRUCTIONS WITH
1340; A 'BIT' INSTRUCTION SAVES A BYTE, BUT MUST
1360; BE USED WITH CARE
1380BITOP = $2C ;OPCODE OF BIT abs
1400;
1420; The first half is basically identical to the
1440; original OSI version.
1460ORG = $FD00 ; CHANGE ORG TO PUT THIS ROUTINE IN RAM
```

```
1480*=ORG
1500        TXA ;SAVE REGISTERS
1520        PHA
1540        TYA
1560        PHA
1580STARTL  LDA #1 ;FOR FIRST ROW
1600NEXROW  JSR PUTROW
1620        JSR GETCOL
1640        BNE FOUNDS
1660TNEXT   ASL A ;FOR NEXT COLUMN
1680        BNE NEXROW
1700        BEQ CLRPRV ;CLEAR ANY PREVIOUS KEY INDICATO
1720FOUNDS  LSR A
1740        BCC REALKY ;NOT JUST SHIFT/CONTROL KEYS
1760        ROL A ;RESTORE A
1780        CPX #$21 ;SHIFT LOCK AND ESC
1800        BNE TNEXT
1820        LDA #$1B ;ESC
1840        BNE COMPRV
1860REALKY  JSR BITNUM ;FIND BIT NUMBER OF ROW
1880        TYA
1900        STA RESULT
1920        ASL A
1940        ASL A
1960        ASL A
1980        SEC
2000        SBC RESULT
2020        STA RESULT ;ROW NUMBER TIMES 7
2040        TXA
2060        LSR A
2080        JSR BITNUM ;FIND COLUMN NUMBER
2100        BNE CLRPRV
2120        CLC
2140        TYA
2160        ADC RESULT
2180        TAY
2200        LDA TABLE,Y
2220        AND #$7F
2240;
2260; Compare this key with the previous iteration.
2280COMPRV  CMP RAWKEY
2300        BNE DIFKEY
2320        DEC ITCNT
2340        BEQ STITER
2360        LDX #4
2380        JSR W1250U ;WAIT 4*1250 MICRO SECONDS
2400        JMP STARTL
2420CLRPRV  LDA #0
2440        STA OLDKEY
2460DIFKEY  STA RAWKEY
2480        LDA #2 ;ITERATION COUNT WHEN NEW KEY HIT
2500        STA ITCNT
2520        BNE STARTL ;BRANCH ALWAYS, START AGAIN
2540;
2560STITER  LDX #$96 ;ITERATION COUNT TO START AUTO REP
2580        CMP OLDKEY
2600        BNE STOIT
2620        LDX #$14 ;AUTO REPEAT ITERATION COUNT
2640STOIT   STX ITCNT
2660        STA OLDKEY
2680;
2700; NOW COMES THE NEW ALGORITHM
2720;
2740        LDA #1
2760        JSR PUTROW
2780        JSR GETCOL ;GET COLUMN IN X
2800;
2820; CHECK IF KEY IS BETWEEN $21 AND $3F (! AND ?)
2840        LDA RAWKEY
2860        CMP #$21
2880        BCC CNTLKY
2900        CMP #$7F
2920        BEQ CNTLKY
2940        AND #$40
2960        BEQ CHSH16  ; BRANCH IF IT IS A NUMBER
2980;
3000; NOW HANDLE LETTERS
3020LETTER  TXA
3040        AND #$40 ;CHECK FOR CONTROL
3060        BNE E64
3080        TXA
3100        AND #7   ;SEE IF THERE ARE ANY SHIFTS
3120        BEQ E32  ;NO SHIFTS MEANS LOWER CASE
3140        LDX RAWKEY ;CHECK FOR SPECIALS (K TO P)
3160        CPX #$51 ;(='Q')
3180        BCS NOTSPE
3200        CPX #$4B ;(='K')
3220        BCC NOTSPE
3240;
3260; THE KEY IS A LETTER BETWEEN 'K' AND 'P'
3280;
3300        CMP #2  ;LOOK FOR RIGHT SHIFT ONLY
3320        BEQ E16
3340        LSR A
3360        BEQ E0  ;SHIFT-LOCK ONLY
3380        BCS E16 ;SHIFT LOCK AND SHIFT
3400;               ELSE NOT SHIFT LOCK AND LEFT SHIFT
3420NOTSPE = * ; 'NORMAL' LETTERS AND SOME SHIFTS
3440CNTLKY = * ; CR,LF etc IGNORE SHIFTS HENCE EOR #0
3460E0      LDA #0
3480        BEQ COMPAN   ;COMPUTE ANSWER
3500CHSH16  TXA
3520        AND #6 ;MASK OUT EVERYTHING EXCEPT LS/RS
3540        BEQ E0 ;NEITHER LEFT NOR RIGHT SHIFT
3560E16     LDA #$10
3580        .BYT BITOP
3600E32     LDA #$20
3620        .BYT BITOP
3640E64     LDA #$40
3660COMPAN  EOR RAWKEY
3680        STA RESULT  ;THIS IS THE FINAL ANSWER
3700        PLA
3720        TAY  ;RESTORE REGISTERS
3740        PLA
3760        TAX
3780        LDA RESULT ;RETRIEVE CHARACTER
3800        RTS
```

CIRCUIT DESIGN

by Ron Cork

Here are two short routines for those of you into circuit design, doing an electronics course, or just like to collect such things to fill up file space. They were adapted from a text book and will calculate both the DC parameters and the AC small signal values, (hybrid equivalent circuit), of a transistor amplifier circuit and a JFET circuit. The programs also display, (screen only), a fully labelled circuit diagram, showing all the correct parameters in their correct places. The screen pokes were built up over a few weeks and are pretty raw so, unfortunately, are not directly adaptable to the various screen formats currently in vogue and I lack both the time and incentive to clean them up, (I count myself lucky to have found the time to even write the routines in the first place).

```
1 REM  The only disk commands in this program are 'DISK!"CL'
2 REM  for screen clear and #DV for device number.
3 REM  These can be deleted without any ill effects.
4 REM
5 REM  All values over 999 are entered and displayed in
6 REM  computer exponent jargon,i.e.  1.5E3  for 1.5K ohms.
7 REM  and  2.1762342E-03  for xxxx milli amps.
8 REM
9 REM  The screen pokes are for 64x64 screen - C4P
10 :
11 DISK!"CL"
20 PRINTTAB(12);"(O) Copyleft ---  R.K.Cork  ---":PRINT:PRINT:PRINT
30 PRINTTAB(5);"These routines can be copied without the explicit
40 PRINTTAB(9);"sanction of the above copyleft holder."
50 FORQ=1TO5:PRINT:NEXT:FORQ=1TO3000:NEXT
60 DISK!"CL
70 PRINTTAB(15)"DC and AC Hybrid Circuit Analysis
80 PRINTTAB(15)"---------------------------------
90 FORQ=1TO10:PRINT:NEXT
100 PRINT"DC circuit analysis.............(1)":PRINT
110 PRINT"AC hybrid circuit analysis.......(2)":PRINT
120 PRINT"End program & return to BEXEC*...(3)
130 PRINT:INPUT"Your choice ";A
140 ONAGOTO160,540,1040
150 PRINT:PRINT
160 DISK!"CL"
170 PRINTTAB(8)"----------------------------------------":PRINT
180 PRINTTAB(8)": This program calculates the DC bias :":PRINT
190 PRINTTAB(8)":    for a voltage-divider biased,     :":PRINT
200 PRINTTAB(8)":     emitter-stabilized circuit.      :":PRINT
210 PRINTTAB(8)"----------------------------------------":PRINT
220 FORQ=1TO5:PRINT:NEXT
230 PRINT"Use Rb2 = 1E30 for Rb2 = infinity.":PRINT
240 PRINT:PRINT"Input the appropriate circuit values.":PRINT
250 PRINT:PRINT:PRINT"Voltage-divider bias resistors :-":PRINT
260 INPUT"Rb1 =";R1
270 INPUT"Rb2 =";R2
280 INPUT"Emitter resistor - RE =";RE
290 INPUT"Collector resistor - RC =";RC
300 INPUT"DC supply volts - Vcc =";SV
310 INPUT"Beta (HFE) =";B
320 DISK!"CL":GOSUB1400
330 PRINT#DV
340 PRINT#DV,"The resulting DC bias currents are :-"
```

```
350 PRINT#DV,"------------------------------------":PRINT#DV
360 VT=R2*SV/(R1+R2):RT=(R2/(R1+R2))*R1
370 IB=(VT-0.7)/(RT+B*RE)
380 IC=B*IB
390 IE=(B+1)*IB
400 PRINT#DV,"IB  = ";IB;"amps":PRINT#DV
410 PRINT#DV,"IC  = ";IC;"amps":PRINT#DV
420 PRINT#DV,"IE  = ";IE;"amps":PRINT#DV:PRINT#DV
430 PRINT#DV,"The DC voltages are :-"
440 PRINT#DV,"--------------------":PRINT#DV
450 VE=IE*RE:VB=VE+0.7:VC=SV-IC*RC:VX=VC-VE:GOSUB1480
460 PRINT#DV,"VB  = ";VB;"volts":PRINT#DV
470 PRINT#DV,"VE  = ";VE;"volts":PRINT#DV
480 PRINT#DV,"VC  = ";VC;"volts":PRINT#DV
490 PRINT#DV,"VCE = ";VX;"volts"
500 GOSUB1060
510 PRINT:PRINT:INPUT"Another ";Y$
520 IFY$="Y"THEN160
530 GOTO60
540 DISK!"CL
550 PRINTTAB(8)"---------------------------------------------
560 PRINTTAB(8)": This program calculates the AC parameters :":PRINT
570 PRINTTAB(8)": of a circuit using the Hybrid Equivalent  :":PRINT
580 PRINTTAB(8)":             circuit model.                :":PRINT
590 PRINTTAB(8)"----------------------------------------------
600 FORQ=1TO10:PRINT:NEXT
610 PRINT"Enter the following circuit component values :-
620 PRINT:PRINT:PRINT
630 INPUT"RB1 = ";R1:PRINT
640 INPUT"RB2 = (use 1E30 if infinite) ";R2:PRINT
650 INPUT"RC  = ";RC:PRINT
660 INPUT"RE  = (unbypassed value only) ";RE:PRINT
670 INPUT"Load resistor RL = (use 1E30 if output open) ";RL
680 PRINT
690 PRINT"Now enter values for the transistor hybrid ";
700 PRINT"equivalent circuit :-"
710 PRINT:INPUT"Hie = ";HI
720 INPUT"Hfe = ";HF
730 INPUT"Hoe = ";HO
740 INPUT"Hre = ";HR
750 :
760 REM  AC circuit calculations
770 :
780 R3=R1*(R2/(R1+R2))
790 RP=RC*(RL/(RC+RL))
800 D=1+HO*RP
810 ZI=HI-HF*HR*RP/D
820 AI=(R3/(R3+ZI))*(HF/D)
830 AV=HF*RP/(HI+(HI*HO-HF*HR)*RP)
840 ZI=R3*(ZI/(R3+ZI))
850 Y2=HO-HF*HR/(HI+RS)
860 IFY2=0THENZ2=1E30:GOTO880
870 Z2=1/Y2
880 ZO=RC*(Z2/(RC+Z2))
890 AP=ABS(AV*AI)
900 DISK!"CL
910 GOSUB1400
920 PRINT#DV:PRINT#DV,"Results of AC circuit analysis are :-"
930 PRINT#DV,"------------------------------------":PRINT#DV
```

Next page please

```
940 FORQ=1TO14:PRINT#DV:NEXT
950 PRINT#DV,"Current Gain     Ai = ";AI
960 PRINT#DV,"Voltage Gain     Av = ";AV
970 PRINT#DV,"Input Impedance  Zi = ";ZI;"ohms"
980 PRINT#DV,"Output Impedance Zo = ";ZO;"ohms"
990 PRINT#DV,"Power Gain       Ap = ";AP:PRINT#DV:PRINT#DV
1000 GOSUB1060
1010 INPUT"Another ";Y$
1020 IFY$="Y"THEN540
1030 GOTO60
1040 DISK!"SE A"
1050 RUN"BEXECX"
1060 REM    Circuit diagram pokes for 64x64 screen
1065 :
1070 P=54378    : REM   Right screen location
1080 POKEP-5,148:POKEP-4,148:POKEP-3,146:POKEP-2,147
1090 POKEP-1,148:POKEP,148:POKEP+1,148:POKEP+2,148:POKEP+3,148
1100 POKEP+4,148:POKEP+5,148:POKEP+6,147
1110 POKEP-58,147:POKEP+70,147:POKEP-57,189:POKEP+71,19
1120 POKEP-63,147:POKEP+65,147:POKEP-127,147:POKEP+129,147
1130 POKEP-190,189:POKEP-254,190:POKEP-318,189:POKEP-382,190
1140 POKEP+194,190:POKEP+258,189:POKEP+322,190:POKEP+386,189
1150 POKEP+450,146:POKEP+514,146:POKEP-446,146:POKEP-510,146
1160 POKEP+136,146:POKEP+200,190:POKEP+264,189:POKEP+328,190
1170 POKEP+392,189:POKEP+456,146:POKEP+520,146:POKEP+583,177
1180 POKEP+584,175:POKEP+577,177:POKEP+578,175:POKEP-6,226
1190 POKEP-574,144:POKEP-573,144:POKEP-572,144:POKEP-571,144
1200 POKEP-570,144:POKEP-569,144:POKEP-568,144:POKEP-567,144
1210 POKEP-566,144:POKEP-565,144:POKEP-564,144:POKEP-563,144
1220 POKEP-562,144:POKEP-561,144:POKEP-559,219:POKEP-557,86
1230 POKEP-556,99:POKEP-555,99:POKEP-320,49:POKEP-321,66
1240 POKEP-322,82:POKEP+318,82:POKEP+319,66:POKEP+320,50
1250 POKEP+330,82:POKEP+331,69:POKEP-120,146:POKEP-184,189
1260 POKEP-248,190:POKEP-312,189:POKEP-376,190:POKEP-440,146
1270 POKEP-504,146:POKEP-310,82:POKEP-309,67:POKEP-119,148
1280 POKEP-118,148:POKEP-117,148:POKEP-116,146:POKEP-115,147
1290 POKEP-114,148:POKEP-113,148:POKEP-112,226:POKEP+9,72
1300 POKEP+10,70:POKEP+11,69:POKEP-60,20:POKEP-251,66:POKEP-252,86
1310 POKEP-124,149:POKEP-380,149:POKEP-444,16:POKEP-8,18
1320 POKEP-9,148:POKEP-11,105:POKEP-12,90:POKEP-110,22:POKEP-108,90
1330 POKEP-107,111:POKEP-177,20:POKEP-241,149:POKEP-370,86
1340 POKEP-369,67:POKEP-497,16:POKEP+202,135:POKEP+203,135
1350 POKEP+204,135:POKEP+205,135:POKEP+206,135:POKEP+207,135
1360 POKEP+271,16:POKEP+335,149:POKEP+463,86:POKEP+464,69
1370 POKEP+591,20:POKEP+143,20:POKEP+78,86:POKEP+79,67:POKEP+80,69
1380 POKEP-49,16
1390 RETURN
1400 DV=2:INPUT"Dump to printer ";Y$
1410 IFLEFT$(Y$,1)="Y"THENDV=4
1420 DISK!"CL":RETURN
1480 VE=INT(VE*100)/100:VB=INT(VB*100)/100
1490 VC=INT(VC*100)/100:VX=INT(VX*100)/100
1500 RETURN
```

The listing of Ron Cork's second program will be in next months newsletter.

ORDER OUT OF KAOS
by Frank Nicholls

Choosing a sorting algorithm involves careful examination of the material to be sorted. Sorting routines vary markedly in speed and the rate of sorting depends on the number of items and the degree of existing order.

If N items are randomly arranged, the speed of a bubble sort varies as N^2, of a Shell-Metzner sort as $N^{1.5}$ and a Quicksort as N x log N. These differences are marked when N is large; for 1000 items the speeds are roughly in the ratio 1 : 4 : 150. These differences fall off for smaller groups, e.g. for 100 items the ratios are 1 : 2 : 20.

However, if the data are almost in order the comparison can change, since the sorting methods vary in the number of swaps and comparisons involved.

I recently used the three sorting techniques to sort 100 3-digit numbers with the following results:

Numbers Arranged:	Quicksort	Shell Sort	Bubble Sort
Randomly	19 sec	30 sec	112 sec
In Order	89 sec	10 sec	69 sec
In Reverse	91 sec	19 sec	179 sec

I then used a split sorting arrangement in which the numbers were sorted in groups of 50 and then merged. The results were:

Numbers Arranged	Quicksort	Shell Sort	Bubble Sort
Randomly	25 sec	20 sec	65 sec
In Order	49 sec	13 sec	28 sec
In Reverse	50 sec	20 sec	90 sec.

Merging the sorted lists involved about 2-3 seconds of these times. It is clear that the split Shell sort can give fast sorting irrespective of the degree of disorder of the list.

DABUG EDITING AND CTRL B FOR THE ASSEMBLER
by John Whitehead with help from David Dodds

Below is the source to enable the Assembler/Editor to be used with a Superboard series 2 in 24X24 or 48X12 with on screen editing.

It may work on other OSIs if line 560 goes to the DABUG initialize routine.

To use the source below, load the Assembler, change the source file workspace start at $12C9 and $12CA to start at $1440 as page 26 of the manual. Type in the source. Assemble it using A and if there are no errors, save it on tape. Then use A3 to put the code into memory. Press the BREAK key, M 1300 G, then answer INIZ? with N. Now try out all the DABUG control keys.

The line feed key is now TAB and E takes you to the monitor. This can be changed in line 630 to go to Exmon in EPROM.

To make a new tape use the token Checksum copier by A. Cashin in conjunction with the article in KAOS Sep.82 about Basic at any Address.

Next page please

```
10   * = $1300
20        JMP DABUG
30     * = $1315
40        .BYT $0A ;line feed
50     * = $1311
60        JSR INPUT
70     * = $1333
80        JMP OUTPUT
90     * = $1391
100  INPUT JSR PUSH
110         LDA #00
120         STA $0E
130         JSR $FFEB
140         PHA
150         LDA $0E
160         BEQ NOBACK
170 SHIFTO  DEC $3F
180         INC $0E
190         BMI SHIFTO
200 NOBACK  PLA
210         TAX
220         JMP PULL
230 OUTPUT  JSR PUSH
240         JSR $FFEE
250         JMP PULL
260 PUSH    PHA
270         TXA
280         PHA
290         LDX #$E0
300 M1      LDA $00,X
310         STA SPACE-$E0,X
320         LDA SPACE+$20-$E0,X
330         STA $00,X
340         INX
350         CPX #$00
360         BNE M1
370         PLA
380         TAX
390         PLA
400         RTS
410 PULL    PHA
420         TXA
430         PHA
440         LDX #$E0
450 M2      LDA $00,X
460         STA SPACE+$20-$E0,X
470         LDA SPACE-$E0,X
480         STA $00,X
490         INX
500         CPX #$00
510         BNE M2
520         PLA
530         TAX
540         PLA
550         RTS
560 DABUG   JSR $FAA1 ;switch to DABUG
570         JSR PULL
580         JMP $1160
590 INOFF   LDA #$BA ;DABUG input off
600         STA $0218
610         LDA #$FF
620         STA $0219
630         JMP $FE00
640 SPACE
650 * = $0670
660         JMP INOFF
670 .END
```

KAOS-WA

Each meeting, our small group seems to be growing and at our May meeting two new members were introduced. At this meeting three members brought their computers along and we were able to compare notes and see how different people have developed their systems. The three systems were all cassette based C1's. Wayne Geary showed us his speech card which he has developed and gave us a demonstration. We were also able to try out a game which he has devised. On Graham Gaiger's system we were able to see BASIC 5 and TOOL-KIT demonstrated. The other system was brought in by Arnold Shepperson.

We also decided to start our own library, and Arnold volunteered to be our librarian. At this point the library contains some past editions of PEEK(65), AARDVARK magazines and a list of tapes in the main library. Any KAOS-WA member may borrow from the library after paying a $5 fee. We have also subscribed to PEEK(65) and these will join the library as they arrive. Members who wish to obtain a magazine from the library can do so at meetings or by phoning Arnold on 322 1388 (WORK ask for WORKSHOP) or 279 7656 (HOME). To administer the library fee Jo Fisher volunteered to be treasurer, so would members who were not at the last meeting but who wish to borrow from the library, please forward $5 to Jo at pay at the next meeting.

Our next meeting is to be on Sunday 17th July on the top floor of Guild House 56 Kishorn St, Mt Pleasant at 2pm, all OSI computers welcome. Peter Hughes has advised me that the following meeting in September will not be able to be held at Guild House, so would members please give some thought to another venue for this one meeting only. See you on the 17th of July.
Gerry Ligtermoet

QUEENSLAND USER GROUP MEETING, 15th MAY 1983

Attendance: 15 members, 3 guests. Computers: 4 working, 2 not

The meeting started at 12 noon. Robin Wells arrived first with his fully built twin disk system. He and Paul Brodie were busy writing a M/C program to enable selection of the various S.E.K. screen formats from a menu. Alan Calvert's S.E.K. had not yet arrived.

Brendan Vowles had also upgraded to twin disks and had planned to give a demo of a music program, but hadn't got it going in time.

Doug Robinson had his cassette based series II along. He had some trouble with Bernie Wills' Tasker RAM board, and as the memory size kept changing on coldstart, it seemed that a 2114 had expired, a regular occurrence.

Tony ?, a friend of Nick Zuryn, had brought along his eprom programmer which wouldn't go. Some of the group hopped in to help, and soon detected that the 8T28s were missing from the Superboard. Doug Robinson was keen to see it going, and tried to talk Tony into using wire links, but at the last moment, he decided to wait and install the chips.

Bob Best discovered to his horror that his C4 had quit. On a break, the screen cleared and a "C" was printed. The machine refused to communicate further. Unless it turned out to be socket problems, I figured the monitor ROM might be stuffed, an ideal excuse for Bob to upgrade to a Dabug!

Much interest was shown in Bert Patterson's very economical 16K RAM board, reviewed in the May KAOS. This expansion board will now go into my system, giving me 24K RAM and, with Bert's Eprom Extender, 18K of ROM.

Trevor Stephenson demonstrated a very useful looking M/C debug routine, which had yet to be completed and documented. Brian Schneider had fitted a disk to his system.

The news that the Electronic Circuit had sold out of their Ohio stock, with twenty Superboards going for between $50 and $100, was met with a stunned silence. I was rather peeved that I had not been told when they were reduced to $150 (and none sold).

John Froggatt arrived a bit late, and he and Brendan stayed until it started to get dark. Finally, the plugs were pulled at 6.10pm.

Ed Richardson.

IS ANYBODY OUT THERE

One of our members, Ken Maclean, who lives in ., would like to contact some radio amateurs. Computers being a bit scarce on the ground around Gulgong, and phone calls being very expensive, Ken would like to use his radio to converse. Ken's call sign is and he operates on the 40 or 80 metre bands.

FOR SALE

C-4P, with 32K, 2MHz, one 8" DSDD Floppy disk drive, 22 diskettes inc. Compdos 1.3, Pascelf, home-grown word processor, games. Modified B&W TV, Microline 80 + 1.5 boxes of paper; 2 joysticks and much documentation. Will separate. $1900 ono. Contact Peter McLennan

INSTRUCTION SUMMARY

MNEMONIC	OPERATION	ADDRESSING MODE																																														PROCESSOR STATUS CODES							
		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND, X)			(IND), Y			Z. PAGE, X			ABS, X			ABS, Y			RELATIVE			INDIRECT			Z. PAGE, Y			BIT ADDRESSING (OP BY BIT #)								7	6	5	4	3	2	1	Ø
		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	Ø	1	2	3	4	5	6	7	N	V	·	B	D	I	Z	C
ADC	A + M + C → A (1) (5)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3				72	5	2												N	V	·	·	·	·	Z	C
AND	A ∧ M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3				32	5	2												N	·	·	·	·	·	Z	·
ASL	C ← [7 Ø] ← Ø				ØE	6	3	Ø6	5	2	ØA	2	1										16	6	2	1E	7	3																					N	·	·	·	·	·	Z	C
BBR[#(Ø-7)]	Branch on M_b = 0 (4)																																								ØF	1F	2F	3F	4F	5F	6F	7F	·	·	·	·	·	·	·	·
BBS[#(Ø-7)]	Branch on M_b = 1 (4)																																								8F	9F	AF	BF	CF	DF	EF	FF	·	·	·	·	·	·	·	·
BCC	Branch on C = Ø (2)																															9Ø	2	2															·	·	·	·	·	·	·	·
BCS	Branch on C = 1 (2)																															BØ	2	2															·	·	·	·	·	·	·	·
BEQ	Branch on Z = 1 (2)																															FØ	2	2															·	·	·	·	·	·	·	·
BIT	A ∧ M	89	2	2	2C	4	3	24	3	2													34	4	2	3C	4	3																					M_7	M_6	·	·	·	·	Z	·
BMI	Branch on N = 1 (2)																															3Ø	2	2															·	·	·	·	·	·	·	·
BNE	Branch on Z = Ø (2)																															DØ	2	2															·	·	·	·	·	·	·	·
BPL	Branch on N = Ø (2)																															1Ø	2	2															·	·	·	·	·	·	·	·
BRA	Branch Alloys (2)																															80	2	2															·	·	·	·	·	·	·	·
BRK	Break (See Fig 1)													00	7	1																																	·	·	·	1	·	1	·	·
BVC	Branch on V = Ø (2)																															5Ø	2	2															·	·	·	·	·	·	·	·
BVS	Branch on V = 1 (2)																															7Ø	2	2															·	·	·	·	·	·	·	·
CLC	Ø → C													18	2	1																																	·	·	·	·	·	·	·	Ø
CLD	Ø → D													D8	2	1																																	·	·	·	·	Ø	·	·	·
CLI	Ø → I													58	2	1																																	·	·	·	·	·	Ø	·	·
CLV	Ø → V													B8	2	1																																	·	Ø	·	·	·	·	·	·
CMP	A − M	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3				D2	5	2												N	·	·	·	·	·	Z	C
CPX	X − M	EØ	2	2	EC	4	3	E4	3	2																																							N	·	·	·	·	·	Z	C
CPY	Y − M	CØ	2	2	CC	4	3	C4	3	2																																							N	·	·	·	·	·	Z	C
DEC	M − 1 → M				CE	6	3	C6	5	2	3A	2	1										D6	6	2	DE	7	3																					N	·	·	·	·	·	Z	·
DEX	X − 1 → X													CA	2	1																																	N	·	·	·	·	·	Z	·
DEY	Y − 1 → Y													88	2	1																																	N	·	·	·	·	·	Z	·
EOR	A⊻M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3				52	5	2												N	·	·	·	·	·	Z	·
INC	M + 1 → M				EE	6	3	E6	5	2	1A	2	1										F6	6	2	FE	7	3																					N	·	·	·	·	·	Z	·
INX	X + 1 → X													E8	2	1																																	N	·	·	·	·	·	Z	·
INY	Y + 1 → Y													C8	2	1																																	N	·	·	·	·	·	Z	·
JMP	Jump to New Loc				4C	3	3										7C	6	3																6C	6	3												·	·	·	·	·	·	·	·
JSR	Jump Sub (See Fig 2)				2Ø	6	3																																										·	·	·	·	·	·	·	·
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3				B2	5	2												N	·	·	·	·	·	Z	·
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2									N	·	·	·	·	·	Z	·
LDY	M → Y (1)	AØ	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3																					N	·	·	·	·	·	Z	·
LSR	Ø → [7 Ø] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3																					Ø	·	·	·	·	·	Z	C
NOP	No Operation													EA	2	1																																	·	·	·	·	·	·	·	·
ORA	A∨M → A	Ø9	2	2	ØD	4	3	Ø5	3	2							Ø1	6	2	11	5	2	15	4	2	1D	4	3	19	4	3				12	5	2												N	·	·	·	·	·	Z	·
PHA	A → Ms S − 1 → S													48	3	1																																	·	·	·	·	·	·	·	·
PHP	P → Ms S − 1 → S													Ø8	3	1																																	·	·	·	·	·	·	·	·
PHX	X → Ms S − 1 → S													DA	3	1																																	·	·	·	·	·	·	·	·
PHY	Y → Ms S − 1 → S													5A	3	1																																	·	·	·	·	·	·	·	·
PLA	S + 1 → S Ms → A													68	4	1																																	N	·	·	·	·	·	Z	·
PLP	S + 1 → S Ms → P													28	4	1																																	(Restored)							
PLX	S + 1 → S Ms → X													FA	4	1																																	·	·	·	·	·	·	·	·
PLY	S + 1 → S Ms → Y													7A	4	1																																	·	·	·	·	·	·	·	·
RMB[#(Ø-7)]	Ø → M_b (5)																																								Ø7	17	27	37	47	57	67	77	·	·	·	·	·	·	·	·
RØL	← [7 Ø] ← [C] ←				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3																					N	·	·	·	·	·	Z	C
RØR	→ [C] → [7 Ø] →				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3																					N	·	·	·	·	·	Z	C
RTI	Rtrn Int (See Fig 1)													4Ø	6	1																																	(Restored)							
RTS	Rtrn Sub (See Fig 2)													6Ø	6	1																																	·	·	·	·	·	·	·	·
SBC	A − M − C → A (1) (5)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3				F2	5	2												N	V	·	·	·	·	Z	(3)
SEC	1 → C													38	2	1																																	·	·	·	·	·	·	·	1
SED	1 → D													F8	2	1																																	·	·	·	·	1	·	·	·
SEI	1 → T													78	2	1																																	·	·	·	·	·	1	·	·
SMB[#(Ø-7)]	1 → M_b (4)																																								87	97	A7	B7	C7	D7	E7	F7	·	·	·	·	·	·	·	·
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3				92	5	2												·	·	·	·	·	·	·	·
STX	X → M				8E	4	3	86	3	2																												96	4	2									·	·	·	·	·	·	·	·
STY	Y → M				8C	4	3	84	3	2													94	4	2																								·	·	·	·	·	·	·	·
STZ	0 → M				9C	4	3	64	3	2													74	4	2	9E	5	3																					·	·	·	·	·	·	·	·
TAX	A → X													AA	2	1																																	N	·	·	·	·	·	Z	·
TAY	A → Y													A8	2	1																																	N	·	·	·	·	·	Z	·
TRB	Ā∧M → M				1C	6	3	14	5	2																																							M_7	M_6	·	·	·	·	Z	·
TSB	A∨M → M				ØC	6	3	Ø4	5	2																																							M_7	M_6	·	·	·	·	Z	·
TSX	S → X													BA	2	1																																	N	·	·	·	·	·	Z	·
TXA	X → A													8A	2	1																																	N	·	·	·	·	·	Z	·
TXS	X → S													9A	2	1																																	·	·	·	·	·	·	·	·
TYA	Y → A													98	2	1																																	N	·	·	·	·	·	Z	·

NOTES

1. Add 1 to N if page boundary is crossed
2. Add 1 to N if branch occurs to same page
   Add 2 to N if branch occurs to different page
3. Carry not = Borrow
4. Effects 8-bit data field of the specified zero page address.
5. Add 1 to N if in Decimal Mode

LEGEND

X	-	Index X
Y	=	Index Y
A	=	Accumulator
M		Memory per effective address
M_S		Memory per stack pointer
M_b	-	Selecter zero page memory bit
M_7	=	Memory Bit 7
M_6	-	Memory Bit 6
+	=	Add
−	=	Subtract
∧	=	And
∨	=	Or
⊻	=	Exclusive or
N	=	Number of cycles
#	=	Number of Bytes